2. Ecommerce Platform Search Function

Asymptotic Notation and Algorithm Analysis:

- **Big O Notation**:
  - Big O notation (often denoted as O(n)) describes the upper bound of an algorithm's running time as the input size grows.
  - It helps us analyze how an algorithm performs in terms of time complexity.
  - For example, when we say an algorithm has a time complexity of O(n), it means that its execution time grows linearly with the input size.
  - Big O notation is essential for understanding worst-case scenarios.
- **Best, Average, and Worst-Case Scenarios for Search Operations**:
  - In search algorithms:
    - **Best Case**: Represents the minimum time required for a successful search (e.g., finding the desired item on the first try).
    - **Average Case**: Reflects the expected time considering various inputs (e.g., random data distribution).
    - **Worst Case**: Indicates the maximum time needed for a search (e.g., when the desired item is at the end of the list or not present).
  - Linear search and binary search exhibit different behaviors:
    - **Linear Search**:
      - Best Case: O(1) (when the target item is the first element).
      - Average Case: O(n/2) (assuming random data distribution).
      - Worst Case: O(n) (when the target item is at the end or not found).
    - **Binary Search**:
      - Best Case: O(1) (when the target item is the middle element).
      - Average Case: O(log n) (assuming a sorted array).
      - Worst Case: O(log n) (same as average case).
  - Binary search significantly outperforms linear search for large datasets due to its logarithmic time complexity.

Setup And Implementation

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
```

```
 * and open the template in the editor.
 */
package EcommercePlatformSearchFunction;

/**
 *
 * @author Aishwarya
 */
import java.util.Arrays;
import java.util.Scanner;
class Product {
    private int productId;
    private String productName;
    private String category;

    public Product(int productId, String productName, String category) {
        this.productId = productId;
        this.productName = productName;
        this.category = category;
    }

    public int getProductId() {
        return productId;
    }

    public String getProductName() {
        return productName;
    }
```

```
    public String getCategory() {
        return category;
    }

    @Override
    public String toString() {
        return "Product ID: " + productId + ", Name: " + productName + ", Category: " + 
category;
    }
}

public class ECommerceSearch {
    private Product[] products;
    private int productCount;

    public ECommerceSearch(int capacity) {
        products = new Product[capacity];
        productCount = 0;
    }

    public void addProduct(Product product) {
        if (productCount < products.length) {
            products[productCount] = product;
            productCount++;
            System.out.println("Product added successfully.");
        } else {
            System.out.println("Product array is full. Cannot add more products.");
        }
    }
```

```java
public Product linearSearch(int productId) {
    for (int i = 0; i < productCount; i++) {
        if (products[i].getProductId() == productId) {
            return products[i];
        }
    }
    return null;
}

public Product binarySearch(int productId) {
    int left = 0;
    int right = productCount - 1;

    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (products[mid].getProductId() == productId) {
            return products[mid];
        }
        if (products[mid].getProductId() < productId) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    return null;
}

public void sortProducts() {
```

```java
 Arrays.sort(products, 0, productCount, (p1, p2) -> Integer.compare(p1.getProductId(),
p2.getProductId()));
    }


    public static void main(String[] args) {
        ECommerceSearch ecs = new ECommerceSearch(10);
        Scanner scanner = new Scanner(System.in);


        while (true) {
            System.out.println("\nE-Commerce Platform Search Functionality");
            System.out.println("1. Add Product");
            System.out.println("2. Linear Search Product");
            System.out.println("3. Binary Search Product");
            System.out.println("4. Exit");
            System.out.print("Choose an option: ");
            int choice = scanner.nextInt();


            switch (choice) {
                case 1:
                    System.out.print("Enter Product ID: ");
                    int productId = scanner.nextInt();
                    scanner.nextLine();
                    System.out.print("Enter Product Name: ");
                    String productName = scanner.nextLine();
                    System.out.print("Enter Product Category: ");
                    String category = scanner.nextLine();
                    Product newProduct = new Product(productId, productName, category);
                    ecs.addProduct(newProduct);
                    break;
```

```
case 2:
    System.out.print("Enter Product ID to search (Linear Search): ");
    productId = scanner.nextInt();
    Product foundProductLinear = ecs.linearSearch(productId);
    if (foundProductLinear != null) {
        System.out.println("Found Product: " + foundProductLinear);
    } else {
        System.out.println("Product not found.");
    }
    break;

case 3:
    ecs.sortProducts();
    System.out.print("Enter Product ID to search (Binary Search): ");
    productId = scanner.nextInt();
    Product foundProductBinary = ecs.binarySearch(productId);
    if (foundProductBinary != null) {
        System.out.println("Found Product: " + foundProductBinary);
    } else {
        System.out.println("Product not found.");
    }
    break;

case 4:
    System.out.println("Exiting...");
    scanner.close();
    return;

default:
```

```java
                System.out.println("Invalid option. Please try again.");
            }
        }
    }
}
```

<u>Implementation and Platform Suitability:</u>

- **Product Class Attributes**:
  - Create a Product class with attributes like productId, productName, and category.
- **Linear Search**:
  - Linear search iterates through each element in the array until it finds the desired item or reaches the end. It’s simple but inefficient for large datasets.
  - Simple to implement but inefficient for large inventories.
  - Suitable for small datasets or unsorted lists.
  - Store products in an array and iterate through each element.
  - Best Case: O(1) (when the target item is the first element).
  - Average Case: O(n/2) (assuming random data distribution).
  - Worst Case: O(n) (when the target item is at the end or not found).
- **Binary Search**:
  - Binary search divides the array in half at each step, narrowing down the search range. It’s highly efficient for sorted arrays and significantly outperforms linear search for large inventories.
  - Requires a sorted array.
  - Efficient for large inventories.
  - Divide and conquer approach.
  - Choose binary search if your platform deals with substantial product catalogs.
  - Consider using a balanced binary search tree (e.g., AVL tree) for even better performance.
  - Best Case: O(1) (when the target item is the middle element).
  - Average Case: O(log n) (assuming a sorted array).
  - Worst Case: O(log n) (same as average case).